



**Embedded Linux
Conference**
North America

Toolchains in the New Era

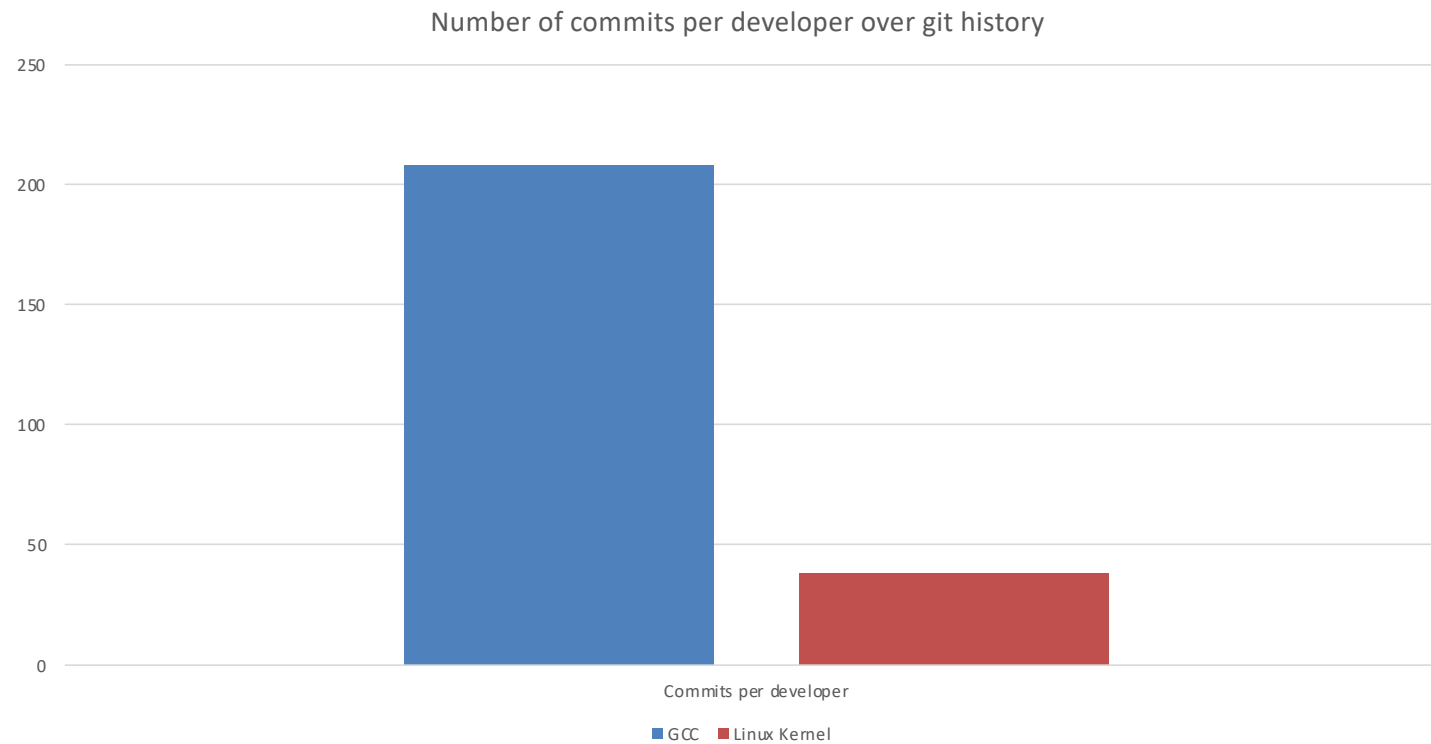
How to Update Safely

SW Eng. Victor Rodriguez

#lfelc @twitterhandle



*GNU compiler community do a lot of work per developer
How much in terms of commits per developer?*



* Information taken from [gcc](#) and [linux](#) githubs repos

Agenda

- What's new?
 - Security
 - Profiling
 - New instructions
- How to manage the new toolchain?
 - Build and install
 - System rebuild
 - Common problems
 - How to fix them?

GCC new features

(security)



[This Photo](#) by Unknown Author is licensed under [CC BY-ND](#)

fanalyzer

- This option enables a static analysis of the program flow that looks for paths through the code, and issues warnings for problems found on them.
- This analysis is much more expensive than other GCC warnings.
- Enabling this option effectively enables [some](#) of these warnings:

Wanalyzer-double-fclose

Wanalyzer-double-free

Wanalyzer-exposure-through-output-file

Wanalyzer-file-leak

Wanalyzer-free-of-non-heap

Wanalyzer-malloc-leak



Examples of -fanalyzer (-Wno-analyzer-double-free)

```
int main(){
    void *ptr;
    free(ptr);
    free(ptr);
    return 0;
}
```

This diagnostic warns for paths through the code in which a pointer can have free called on it more than once.

```

root@159m0rkstation: /dev01/hobbies/c_programming_exercises/lanalyzer # make
gcc test.c -fanalyzer
test.c: In function 'main':
test.c:7:2: warning: double-'free' of 'ptr' [CWE-415] [-Wanalyzer-double-free]
    7 |     free(ptr);
      |     ^~~~~~
'main': events 1-2
|
|
|
6 |     free(ptr);
  |     ^~~~~~
  |
  | (1) first 'free' here
7 |     free(ptr);
  |     ~~~~~~
  |
  |
  | (2) second 'free' here; first 'free' was at (1)

```

Examples of `-fanalyzer` (`-Wno-analyzer-malloc-leak`)

This diagnostic warns for paths through the code in which a pointer allocated via `malloc` is leaked.

```
void test(const char *filename) {  
    FILE *f = fopen(filename, "r");  
    void *p = malloc(1024);  
    /* do stuff */  
}
```



```
test_2.c: In function 'test':  
test_2.c:8:1: warning: leak of 'p' [CWE-401] [-Wanalyzer-malloc-leak]  
8 | }  
  | ^  
  |  
  | test: events 1-2  
6 |     void *p = malloc(1024);  
  |                   ^~~~~~  
  |                   (1) allocated here  
7 |     /* do stuff */  
8 | }  
  | ^  
  | (2) 'p' leaks here; was allocated at (1)  
  
test_2.c:8:1: warning: leak of FILE 'f' [CWE-775] [-Wanalyzer-file-leak]  
8 | }  
  | ^  
  |  
  | test: events 1-2  
5 |     FILE *f = fopen(filename, "r");  
  |                   ^~~~~~  
  |                   (1) opened here  
8 | }  
  | ^  
  | (2) 'f' leaks here; was opened at (1)
```

Examples of `-fanalyzer` (`-Wanalyzer-unsafe-call-within-signal-handler`)

Dangerous `fprintf`

- `fprintf` can be a security vulnerability.
- The problem is that `fprintf` determines how many arguments it should get by examining the format string.
- If the format string doesn't agree with the actual arguments, you have undefined behavior which can manifest as a security vulnerability.
- format string exploits: the attacker can perform writes to arbitrary memory addresses.

```
#include<stdio.h>
int main(int argc, char** argv) {
    char buffer[100];
    strncpy(buffer, argv[1], 100);
    printf(buffer);
    return 0;
}
```



```
$ ./a.out "AAAA%p %p %p %p %p %p %p %p %p"
AAAA0xffffdde8 0x64 0xf7ec1289 0xffffdbef 0xffffdbef (nil) 0xffffdcd4 0xffffdc74 (nil) 0x41414141
```

Credits to <https://developers.redhat.com/blog/2020/03/26/static-analysis-in-gcc-10/>

Examples of `-fanalyzer` (`-Wanalyzer-unsafe-call-within-signal-handler`)

```
#include <stdio.h>
#include <signal.h>

void custom_logger(const char *msg)
{
    fprintf(stderr, "LOG: %s", msg);
}

static void handler(int signum)
{
    custom_logger("got signal");
}

int main(int argc, const char *argv)
{
    custom_logger("started");
    signal(SIGINT, handler);
    return 0;
}
```

```
test_3.c: In function 'custom_logger':
test_3.c:7:3: warning: call to 'fprintf' from within signal handler [CWE-479] [
7 |     fprintf(stderr, "LOG: %s", msg);
  |     ^~~~~~
'main': events 1-2
15 | int main(int argc, const char *argv)
   | ^----
   | (1) entry to 'main'
.....
18 |     signal(SIGINT, handler);
   |     ~~~~~
   | (2) registering 'handler' as signal handler
event 3
cc1:
(3): later on, when the signal is delivered to the process
+--> 'handler': events 4-5
10 | static void handler(int signum)
   | ^-----
   | (4) entry to 'handler'
11 | {
12 |     custom_logger("got signal");
   |     ~~~~~
   | (5) calling 'custom_logger' from 'handler'
+--> 'custom_logger': events 6-7
5 | void custom_logger(const char *msg)
  | ^-----
  | (6) entry to 'custom_logger'
6 | {
7 |     fprintf(stderr, "LOG: %s", msg);
  |     ~~~~~
  | (7) call to 'fprintf' from within signal handler
```

Credits to
<https://developers.redhat.com/blog/2020/03/26/static-analysis-in-gcc-10/>



Extra options

 -fdiagnostics-path-format=separate-events

 -fdiagnostics-format=json

```
gcc test.c -fanalyzer
test.c: In function 'main':
test.c:7:2: warning: double-'free' of 'ptr' [CWE-415] [-Wdouble-free]
   7 |   free(ptr);
     |   ^~~~~~
'main': events 1-2
   6 |   free(ptr);
     |   ^~~~~~
     |   (1) first 'free' here
   7 |   free(ptr);
     |   ^~~~~~
     |   (2) second 'free' here; first 'free' was at
```



```
{
  kind: warning,
  locations: [
    {
      finish: {
        line: 7,
        file: test.c,
        column: 10
      },
      caret: {
        line: 7,
        file: test.c,
        column: 2
      }
    }
  ],
  path: [
    {
      location: {
        line: 6,
        file: test.c,
        column: 2
      },
      description: first 'free' here,
      depth: 1,
      function: main
    }
  ]
}
```



Next steps and real example where to use it

- **CVE-2005-1689** (Kerberos computer-network authentication protocol)
 - Double free vulnerability in the `krb5_recvauth` function in MIT Kerberos 5 (krb5) 1.4.1 and earlier allows remote attackers to execute arbitrary code via certain error conditions.



- It correctly identifies the bug with no false positives,
- Need to improve warnings without overwhelming.

Next steps and real example where to use it

```
In file included from ../../../../src/lib/krb5/krb/recvauth.c:31:
../../../../src/lib/krb5/krb/recvauth.c: In function 'recvauth_common':
../../../../src/lib/krb5/krb/../../../../include/k5-int.h:1670:25: warning: double-'free' of
1670 | #define krb5_xfree(val) free((char *) (val))
      |                        ^~~~~~
../../../../src/lib/krb5/krb/recvauth.c:82:6: note: in expansion of macro 'krb5_xfree'
  82 |     krb5_xfree(inbuf.data);
      |     ^~~~~~
'krb5_recvauth_version': events 1-2

  256 |     krb5_recvauth_version(krb5_context context,
      |     ^~~~~~
      | (1) entry to 'krb5_recvauth_version'
.....
  267 |     return recvauth_common (context, auth_context, fd,
      |     ^~~~~~
      | (2) calling 'recvauth_common' from 'krb5_re
  268 |     server, flags, keytab, ticket, version);
      |     ^~~~~~
```

```
Index: lib/krb5/krb/recvauth.c
=====
RCS file: /cvs/krbdev/krb5/src/lib/krb5/krb/recvauth.c,v
retrieving revision 5.38
diff -c -r5.38 recvauth.c
*** lib/krb5/krb/recvauth.c      3 Sep 2002 01:13:47 -0000      5.38
--- lib/krb5/krb/recvauth.c      23 May 2005 23:19:15 -0000
*****
*** 76,82 ****
+         if ((retval = krb5_read_message(context, fd, &inbuf)))
+             return(retval);
+         if (strcmp(inbuf.data, sendauth_version)) {
+             krb5_xfree(inbuf.data);
+             problem = KRB5_SENDAUTH_BADAUTHVERS;
+         }
+         krb5_xfree(inbuf.data);
--- 76,81 ----
*****
*** 90,96 ****
+         if ((retval = krb5_read_message(context, fd, &inbuf)))
+             return(retval);
+         if (appl_version && strcmp(inbuf.data, appl_version)) {
+             krb5_xfree(inbuf.data);
+             if (!problem)
+                 problem = KRB5_SENDAUTH_BADAPPLVERS;
+         }
--- 89,94 ----
```


GCC new features

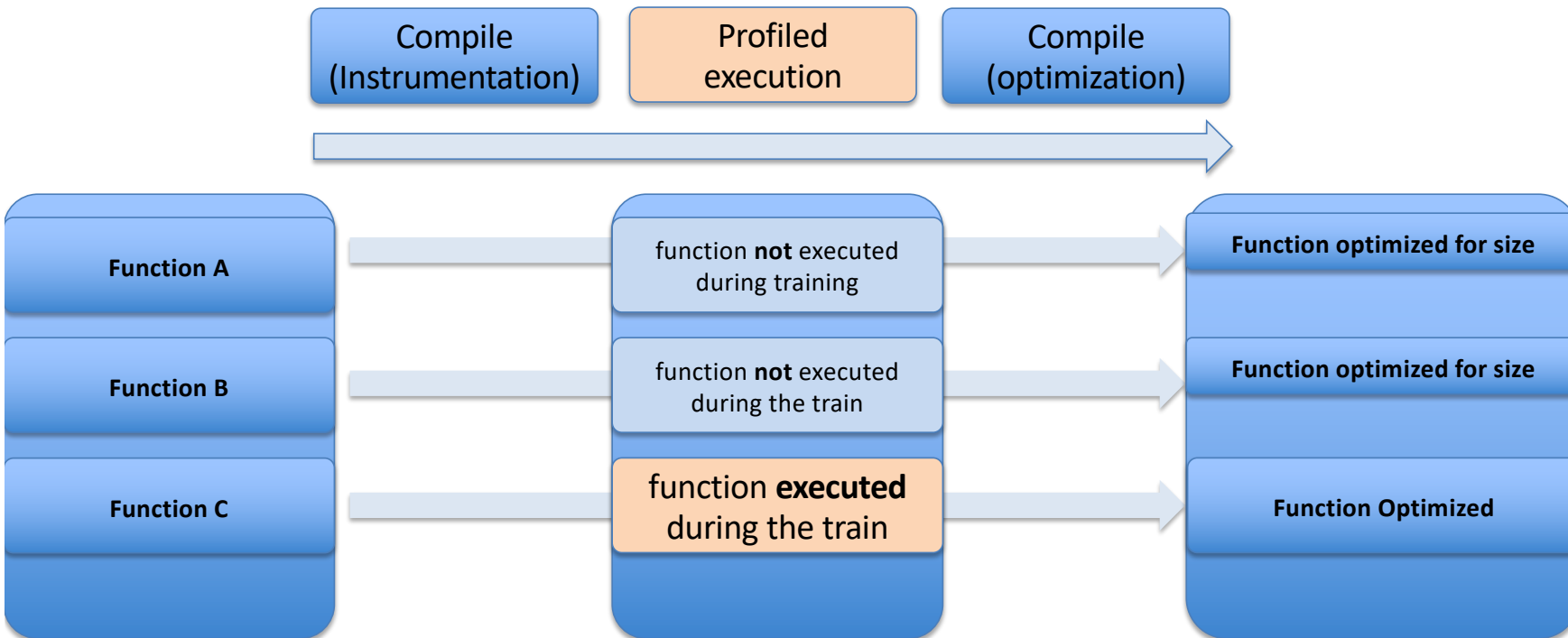
(profiling)



fprofile-partial-training

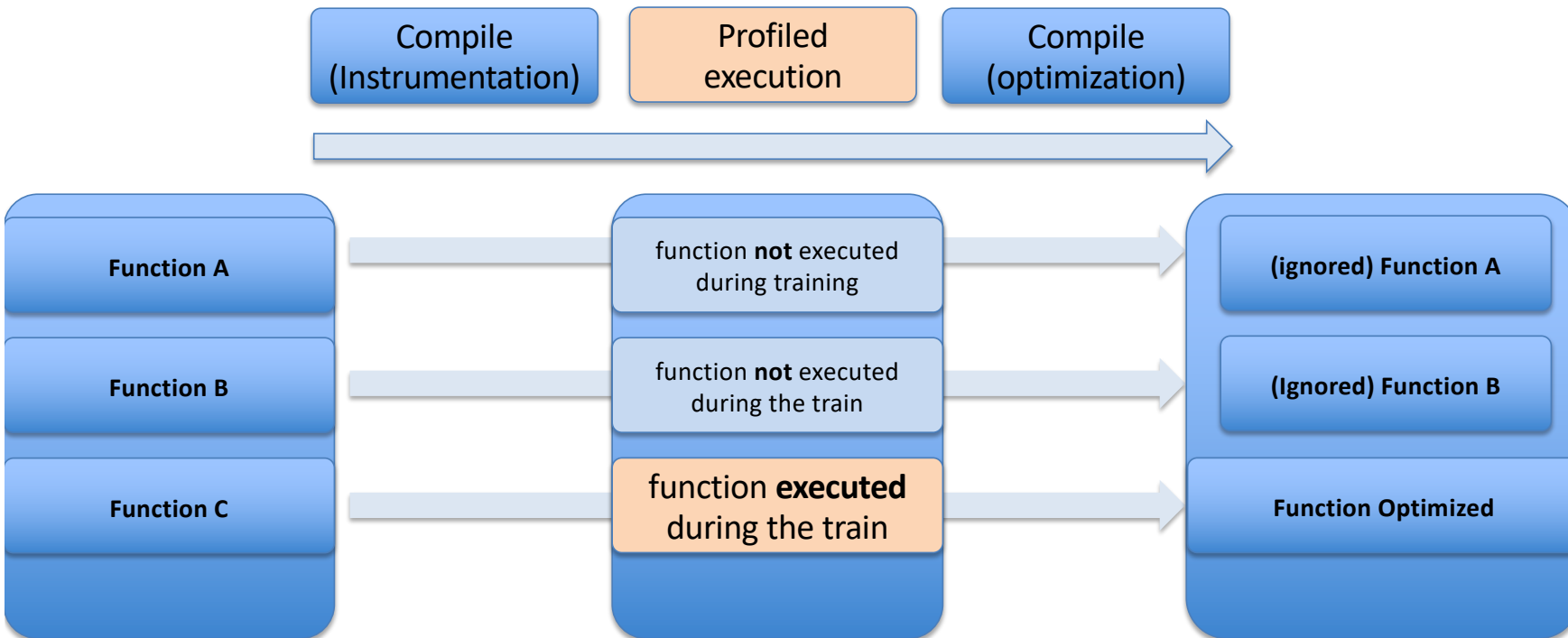
- fprofile-partial-training can now be used to inform the compiler that code paths not covered by the training run should not be optimized for size.

Example w/o fprofile-partial-training



fprofile-partial-training

Example w/ fprofile-partial-training



Profile-guided optimization (PGO) for developers in a hurry

Compilers do performance optimizations by using known heuristics, making most probable guesses about code execution.

- A compiler may decide the branch to take based on where is it located in a loop
- it may choose to inline a function based on its size.



This Photo by Unknown Author is licensed under [CC BY-NC-ND](#)

```
for (int i = 0; i < 100000; ++i) {  
    // Primary loop  
    for (int c = 0; c < arraySize; ++c) {  
        if (data[c] >= 128)  
            sum += data[c];  
    }  
}
```

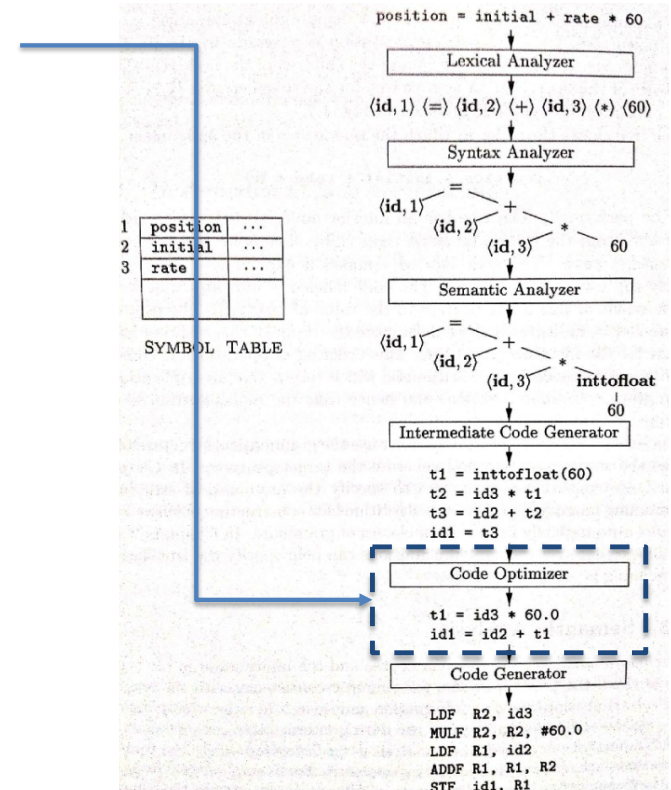


Image from Compilers: Principles, Techniques, and Tools[1] by Alfred V. Aho et al
Reference: <https://developer.ibm.com/technologies/systems/articles/gcc-profile-guided-optimization-to-accelerate-ai-x-applications/>

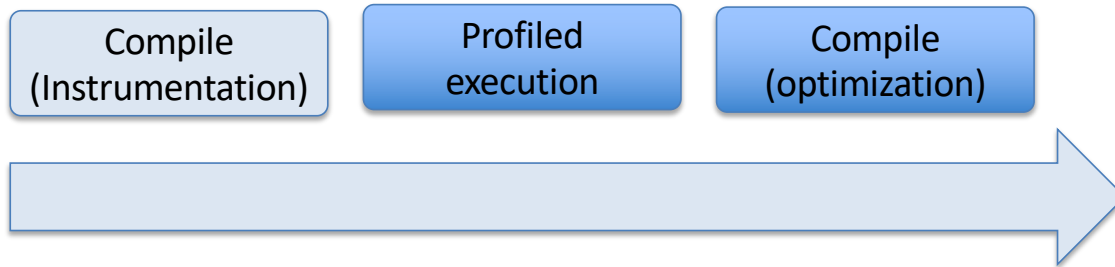
Profile-guided optimization (PGO) for developers in a hurry

- To extract good performance out of a program, it would be nice if programmers could provide hints or annotations to the compiler.
- PGO is a method used by compilers to produce optimal code by using application runtime data.
- Because this data comes directly from the application, the compiler can make much more accurate guesses.



[This Photo](#) by Unknown Author is licensed under [CC BY-SA](#)

How does it work; phases?



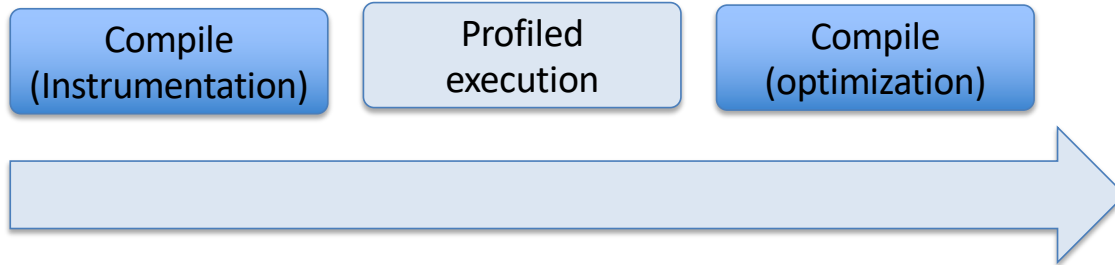
Instrumentation

- Produces an executable program that contains probes in each of the basic blocks of the program.
- Each probe counts the number of times a basic block runs. If the block is a branch, the probe records the direction taken by that branch.

• `$ gcc -fprofile-genrate=<profile_dir> source.c`



How does it work; phases?



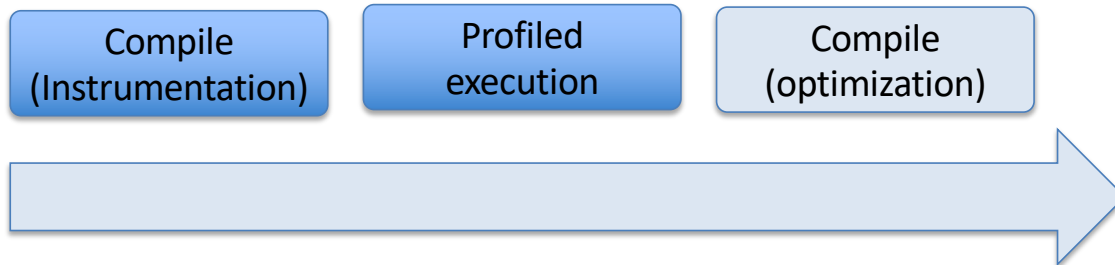
Profiled execution

- When it is executed, the instrumented program generates a data file that contains the execution counts for the specific run of the program



[This Photo](#) by Unknown Author is licensed under [CC BY-SA](#)

How does it work; phases?



Optimization

- Information from the profiled execution of the program is feedback to the compiler. This data is used to make a better estimate of the program's control flow. The compiler uses this information to produce an executable file, relying on this data rather than the static heuristics
- `$ gcc -fprofile-use=<profile_dir> source.c`

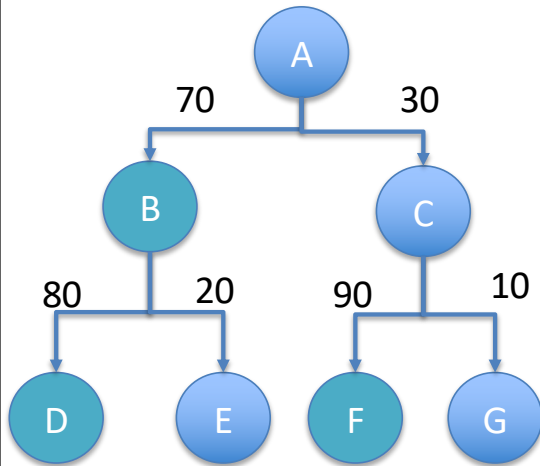


[This Photo](#) by Unknown Author is licensed under [CC BY-NC](#)

What kind of optimizations should I expect?

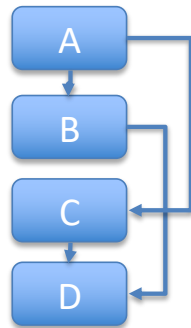
- **Function inlining**

A technique where we inline high frequency functions into one of the calling functions to reduce function call overheads.



- **Block ordering**

A compiler's aim is to achieve call locality or to perform minimum amount of memory operations. Functions which are tightly bound should be collocated to minimize instruction cache fetches.



```
func_a()
{
    .....
    .....
    if (condition = TRUE)
    {
        call_func_b()
    } else {
        call_func_c()
    }
}

func_b()
{
    func_d()
}
```

- **Dead Code Elimination**

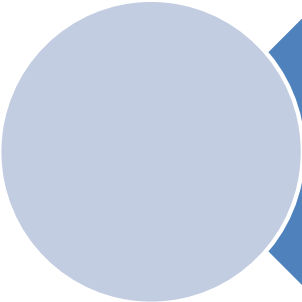
Dead Code Elimination is an optimization that removes code that does not affect the program result. It is important to differentiate:

Dead code: code that is not executed, either because it was never used or because it is unreachable from the rest of the program. It wastes CPU cycles.

Unreachable code: code that is reached but never executed because of logic flow. It is not executed, but with runtime overhead.

```
int foo(int x, int y) {
    int a = x + y;
    a = 1;
    return a;
}
```

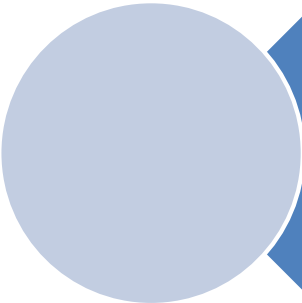
Next steps and real example where to use it



If the application behaves differently for different data sets, it is possible a regression in performance.

Python* already provides an option for this:

```
%configure %python_configure_flags --enable-optimizations  
make profile-opt % {?_smp_mflags}  
%make_install
```



For large applications that are run frequently, it is observed that PGOs can prove to be a significant source of performance.

build

Run
benchmark

Re compile



More info at [Clear Linux python spec](#)

*Other names and brands may be claimed as the property of others.



#felc

And if them profiled execution is in parallel ?

fprofile-reproducible

Control the level of reproducibility of the profile gathered by `-fprofile-generate`.

- With `-fprofile-reproducibility=serial` the profile gathered by `-fprofile-generate` is reproducible provided the trained program behaves the same at each invocation of the training run,



[This Photo](#) by Unknown Author is licensed under [CC BY-ND](#)



And if them profiled execution is in parallel ?

fprofile-reproducible

- With `-fprofile-reproducibility=parallel-runs` collected profile stays reproducible regardless of the order of streaming of the data into gcda files.
- This setting makes it possible to run multiple instances of instrumented programs in parallel (such as with `make -j`).
- This reduces the quality of gathered data, in particular of indirect call profiling.



[This Photo](#) by Unknown Author is licensed under [CC BY-SA](#)

GCC new features

(code health)



[This Photo](#) by Unknown Author is licensed under [CC BY-NC-ND](#)

fno-common: Force good coding

- GCC now defaults to -fno-common. As a result, global variable accesses are more efficient on various targets.
- In C, global variables with multiple tentative definitions now result in linker errors.
- With -fcommon such definitions are silently merged during linking.

gcc will reject multiple definition of global variables starting from gcc-10:

```
//a.c
int a = 42;

// main.c
int a;
int main(){}

```

The fix (source changes, preferred)

The fix is simple: explicitly mark declarations as such and avoid multiple definitions:

```
//a.c
int a = 42;

// main.c
extern int a; // was 'int a;'
int main(){}

```

The -fcommon workaround (discouraged)

```
src_configure() {
    # discouraged, source change fix is preferred
    append-cflags -fcommon # https://link/to/upstream/bug/report
    econf ...
}
```

Bug 705764 depends on 399 open bugs:

[view as bug list](#)

705764: [TRACKER] Packages failing with -fno-common

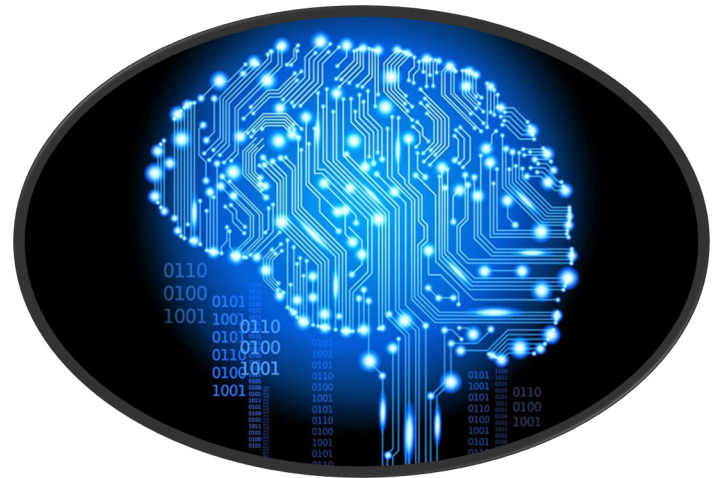
- 706000: net-misc/networkmanager: Fails to compile with -fno-common
- 706350: net-misc/sipsak-0.9.6_p1-r2 : fails to build with -fno-common or gcc-10
- 706396: www-servers/gatling-0.15 : fails to build with -fno-common or gcc-10
- 706398: sys-fabric/infinipath-psm-3.2 : fails to build with -fno-common or gcc-10
- 706414: sci-biology/embassy-phyllipnew-3.69.660 : fails to build with -fno-common or gcc-10
- 706448: games-misc/sdljoytest-1.1102003 : fails to build with -fno-common or gcc-10
- 706450: sci-biology/blat-34-r2 : fails to build with -fno-common or gcc-10
- 706460: media-sound/chordii-4.5.3 : fails to build with -fno-common or gcc-10
- 706516: app-mobilephone/kannel-1.5.0-r4 : fails to build with -fno-common or gcc-10
- 706518: games-misc/typespeed-0.6.5-r1 : fails to build with -fno-common or gcc-10
- 706522: x11-misc/xfractint-20.04_p14 : fails to build with -fno-common or gcc-10
- 706530: sci-biology/mcl-14.137 : fails to build with -fno-common or gcc-10



GCC

new features

(new instructions)



[This Photo](#) by Unknown Author is licensed under [CC BY-NC-ND](#)

New Targets and Target Specific Improvements: IA-32/x86-64

- GCC now supports the Intel® architectures code named Cooper Lake & Tiger Lake through

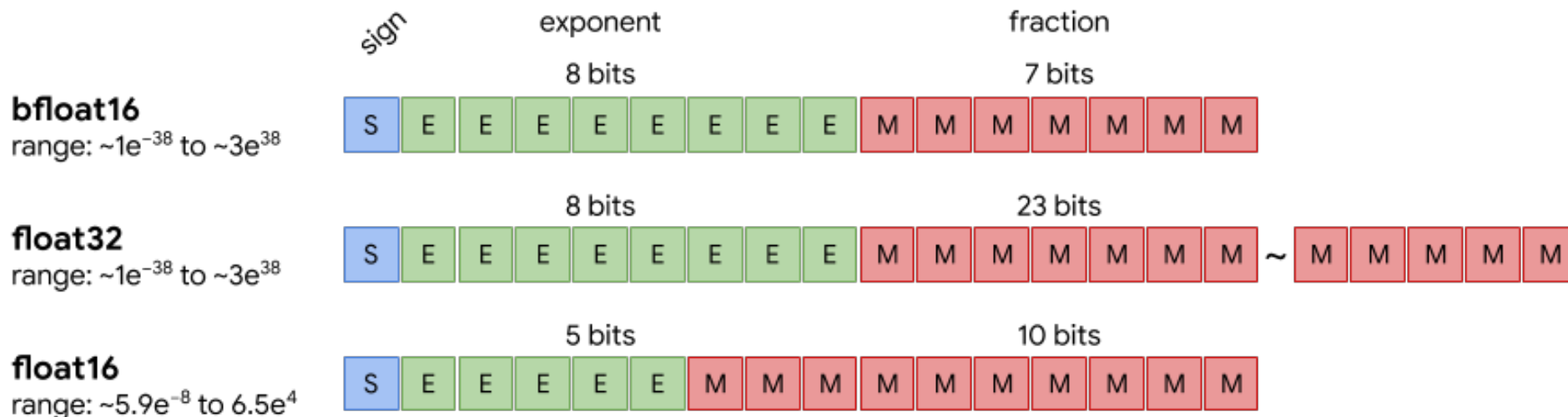
-march=cooperlake/tigerlake.
- If the Cooper Lake switch enables the AVX512BF16 ISA extensions for BFLOAT16

$$\underbrace{6.63}_{\text{Mantissa}} \times \underbrace{10^{-34}}_{\text{Exponent}}$$

$$\underbrace{\overbrace{0}^{\text{sign bit}} \cdot 101010101}_{\text{mantissa}} \times \underbrace{010101}_{\text{exponent}}$$

$$\begin{array}{ccccccc} 4 & 2 & 1 & . & \frac{1}{2} & \frac{1}{4} & \frac{1}{8} \\ 0 & 0 & 1 & . & 0 & 0 & 0 \end{array} = 1 = 0.100\ 0000\ 00 \mid 00\ 0001$$

The dynamic range of bfloat16 is greater than that of fp16



- The bfloat16 range is useful for things like gradients that can be outside the dynamic range of fp16 and thus require loss scaling; bfloat16 can represent such gradients directly.
- In addition, you can use the bfloat16 format to accurately represent all integers $[-256, 256]$, which means you can encode an int8 in bfloat16 without loss of accuracy.

In order to use BF16 efficiently, it must be implemented in hardware in a unified way. The new Cooper Lake instruction provides two flavors of new instructions for this:

- FMA unit with two BF16 input operands and one FP32 input/output operand
- Conversions between FP32 and BF16

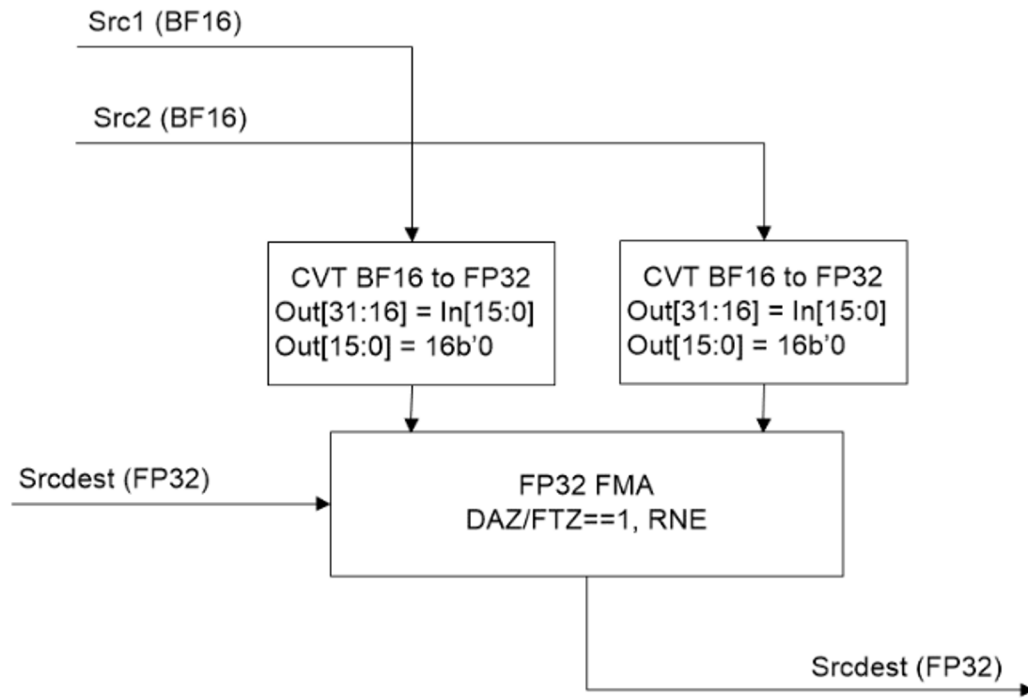
The list of Intel® AVX512 BF16 Vector Neural Network Instructions includes:

VCVTNE2PS2BF16	Conversions	Convert Two Packed Single Data to One Packed BF16 Data
VCVTNEPS2BF16	Conversions	Convert Packed Single Data to Packed BF16 Data
VDPBF16PS	FMA	Dot Product of BF16 Pairs Accumulated into Packed Single Precision

All of them can be executed in 128-bit, 256-bit, or 512-bit mode, so software developers can pick up one of a total of nine versions based on their requirements.

Intel® AVX512 refers to Intel® Advanced Vector Extensions 512

FMA with BFLOAT16 (VDPBF16PS)



BFP10002

This unit takes two BF16 values and multiply-adds (FMA) them as if they would have been extended to full FP32 numbers with the lower 16 bits set to zero.

The BF16*BF16 multiplication is performed without loss of precision; its result is passed to a general FP32 accumulator with the aforementioned settings.

Example with code in C ([full example](#))

```
/* __m128bh _mm_cvtneps_pbh (__m128 a)
Convert packed single-precision (32-bit) floating-point elements in a to
packed BF16 (16-bit) floating-point elements, and store the results in dst.
*/
result = _mm_cvtneps_pbh(A);
```

```
/* __m128bh _mm_cvtne2ps_pbh (__m128 a, __m128 b)
Convert packed single-precision (32-bit) floating-point elements in two
vectors a and b to packed BF16 (16-bit) floating-point elements, and store
the      results in single vector dst. */
result = _mm_cvtne2ps_pbh(A,B);
```

```
/* __m128 _mm_dpbf16_ps (__m128 src, __m128bh a, __m128bh b)
Compute dot-product of BF16 (16-bit) floating-point pairs in a and b,
accumulating the intermediate single-precision (32-bit) floating-point
elements with elements in src, and store the results in dst.
*/
result = _mm_dpbf16_ps(A,A,B);
```

DEMO

```
/* __m128 __mm_dpbf16_ps (__m128 src, __m128bh a, __m128bh b)
Compute dot-product of BF16 (16-bit) floating-point pairs in a and b,
accumulating the intermediate single-precision (32-bit) floating-point
elements with elements in src, and store the results in dst.
*/
result = __mm_dpbf16_ps(A,A,B);
```



```
__m128bh a;
a = 10.000000
__m128bh b;
b = 6.000000
```



```
0.000000
result __mm_dpbf16_ps(A,A,B):
0.000000
70.000000
0.000000
0.000000
```

How to manage the new toolchain?

Build GCC

```
1  tar -xf gcc-${GCC_VERSION}.tar.bz2

   # download the prerequisites
2  cd gcc-${GCC_VERSION}
   ./contrib/download_prerequisites

   # create the build directory
3  cd ..
   mkdir gcc-build
   cd gcc-build
```

Build GCC

```
4  # build
   ../gcc-${GCC_VERSION}/configure \
   --prefix=${INSTALLDIR}          \
   --enable-shared                  \
   --with-system-zlib               \
   --enable-threads=posix           \
   --enable-__cxa_atexit            \
   --enable-clocale=gnu             \
   --enable-languages="c,c++,fortran" \
5  && make \
   && make install
```

Source at : https://github.com/VictorRodriguez/hobbies/blob/master/personal_scripts/build-gcc.sh



Notes

```
#
#
# --enable-shared --enable-threads=posix --enable-__cxa_atexit:
#     These parameters are required to build the C++ libraries to published standards.
#
# --enable-clocale=gnu:
#     This parameter is a failsafe for incomplete locale data.
#
# --disable-multilib:
#     This parameter ensures that files are created for the specific
#     architecture of your computer.
#     This will disable building 32-bit support on 64-bit systems where the
#     32 bit version of libc is not installed and you do not want to go
#     through the trouble of building it. Diagnosis: "Compiler build fails
#     with fatal error: gnu/stubs-32.h: No such file or directory"
#
# --with-system-zlib:
#     Uses the system zlib instead of the bundled one. zlib is used for
#     compressing and uncompressing GCC's intermediate language in LTO (Link
#     Time Optimization) object files.
#
# --enable-languages=all
# --enable-languages=c,c++,fortran,go,objc,obj-c++:
#     This command identifies which languages to build. You may modify this
#     command to remove undesired language
```

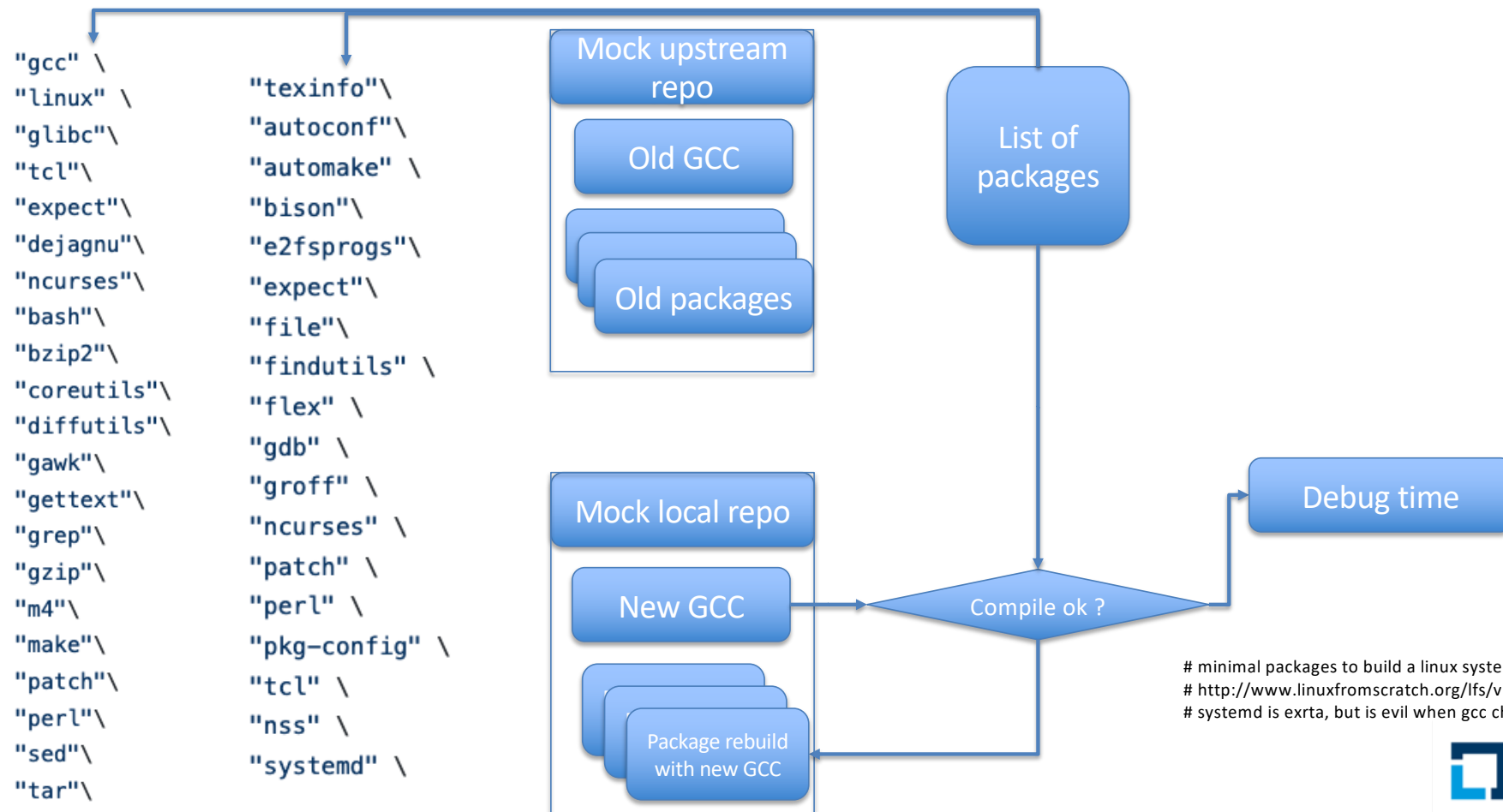


More info at
<http://www.linuxfromscratch.org/lfs/view/development/chapter06/gcc.html>

[This Photo](#) by Unknown Author is licensed under [CC BY-SA](#)



Minimal image packages build process



minimal packages to build a linux system , taken from :
<http://www.linuxfromscratch.org/lfs/view/6.6/index.html>
systemd is exrta, but is evil when gcc change :)



Where can I ask for help?

- GCC 10 provides a [porting to gcc10](#) documentation
- [GCC help ML](#)
- GLIBC ML for packages problems



Image by [Gerd Altmann](#) from [Pixabay](#)

Summary

The toolchain is simply tools to build software (compiler, assembler, linker, libraries, and a few useful utilities).

This is NOT true

*The toolchain is a set of **optimized** tools to build **better** software*

Let's use it



[This Photo](#) by Unknown Author is licensed under [CC BY-SA-NC](#)



Embedded Linux Conference

North America

Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No product or component can be absolutely secure. Check with your system manufacturer or retailer or learn more at [intel.com](https://www.intel.com).

Intel and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

© 2020 Intel Corporation